半導体LSIのデバイスシミュレータ Advance/DESSERTの並列化

桑原　匠史*

Parallelization of Device Simulator Advance/DESSERT for Semiconductor LSI

Takuhito Kuwabara*

プロセス・デバイスシミュレータを用いた半導体数値実験は、半導体設計時に必要不可欠な技術となっており、世界における、半導体製造工場が集約され製造工場を持つ企業が限られる状況において、半導体製造工場を持たない企業などでは、半導体装置の設計現場で特に重要な位置を占めるようになってきている。ここでは、弊社において開発されたデバイスシミュレータ Advance/DESSERT を用いて大規模解析行うために実施した並列化に関する作業を紹介する。

Key word: TCAD、半導体、デバイスシミュレータ、大規模解析

1. はじめに

Advance/DESSERT は、科学技術振興機構殿（JST）の研究成果最適展開支援プログラム（A-STEP）の支援を受け、明治大学と弊社の共同で開発を行ったプロセス・デバイスシミュレータ Advance/TCAD のデバイスシミュレーション部分を受け持つプログラムである。本支援プログラムは、平成２３年に開始され今年度がプログラムの最終年度となっており４月以降にリリースを控えている。是非多くの方々に Advance/TCAD を利用して頂き、日本の半導体産業への貢献を行っていきたいと考えている。

2. 基礎方程式

本プログラムでは、キャリアを連続流体とみなす「流体モデル」を用いて計算対象をモデル化している。「流体モデル」を用いたデバイスシミュレーションの基礎方程式を以下に示す。

半導体内のキャリア（電子、正孔）は、電場により駆動される。その電場は、静電ポテンシャルの勾配より求められ、静電ポテンシャル Ψ は以下の式より求める事ができる。

$$\nabla(\varepsilon\nabla\Psi) = -\rho \qquad (1)$$

ここで、ρ は空間電荷密度、ε は誘電率を表す。次に、電場により駆動される電子と正孔の運動は、以下に示す電流保存則により表す事ができる。

$$\frac{\partial n}{\partial t} - \frac{1}{q}\nabla\cdot\mathbf{J}_n = G - R \qquad (2)$$

$$\frac{\partial p}{\partial t} + \frac{1}{q}\nabla\cdot\mathbf{J}_p = G - R \qquad (3)$$

ここで、n、p は電子、正孔の数密度、q は素電荷、$\mathbf{J}_n$、$\mathbf{J}_p$ は電流密度、G、R は電子の生成率、消滅率を示す。この３つの方程式を有限体積法による陰的解法を用いて解くことにより、半導体内における電子、正孔の振る舞いを数値的に求める。

3. 並列化

3.1. 手法

Advance/DESSERT の並列化を行った際に採用した手法を以下に示す。まず、並列計算のために計算領域の分割を行うと、分割した領域の端の部分を各 CPU コアで保持し、計算中に相互にこの部分の情報を通信を行い補完し合う必要が出て来る。そこで、この重複部分の値を保持する１次元配列を各 CPU コア上に用意し、計算中に必要なときにこの配列から値を呼びだすようにした。ここで、重複部分について一般的には、各 CPU コアで大きさが異なるため、１次元配列のサイズ

*アドバンスソフト株式会社　第１事業部
1st Computational Science and Engineering Group, AdvanceSoft Corporation

も各 CPU コアで異なる。また、この 1 次元配列のインデックスは、1 から始まるようにしているため、情報は本来の節点番号を失った形で保持されることになる（図 1)。よって、重複部分のローカルな番号と本来の節点の番号を結びつける情報を保持する配列が必要となるため、これを用意した。

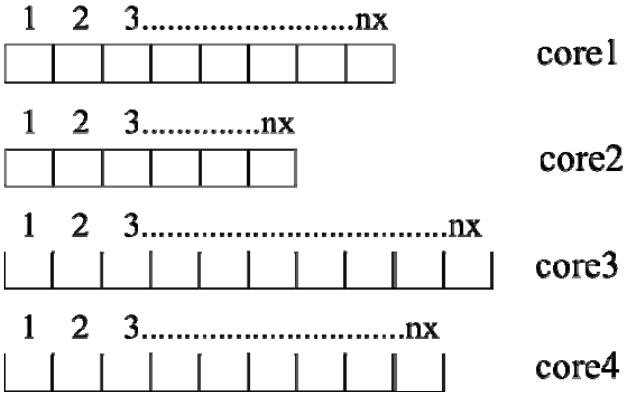

図 1　各 CPU コアにおける重複部分の格納配列のイメージ（4 コアの場合）

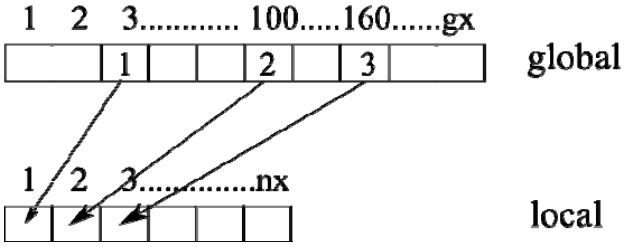

図 2　グローバルな節点番号とローカルな番号の関連づけ

図 2 は、例として、計算中に節点番号 3、100、160 が必要なときにローカルな番号の配列の情報を読みにいく仕組みを示している。まず、各 CPU コアが担当している領域に属する節点番号や計算に必要のない節点番号の global 配列には"-1"などの値を入れておき、それ以外の重複領域の節点番号の global 配列にはローカルな番号を入れておく。プログラム中で計算に必要な節点番号、かつ各 CPU コアの担当外の節点番号が現れたら global 配列から節点番号の配列に格納されているローカルな番号を取り出し、ローカル配列に格納されている物理量を取り出す。このような仕組みを用意することで、領域分割により必要となる重複領域の情報の保持と計算中における値の取得をスムーズに行うことを可能にしている。一方、行列部分の計算については、行列ソルバーライブラリ PETSc[1]を採用している。以下に、非並列計算の結果と、並列計算の結果の比較を示す。例題として MOSFET についての計算を行い、得られたポテンシャル分布を図 3 に示している。左図は非並列計算の結果を示し、右図は並列計算（4CPU コア）の結果を示している。どちらも同じカラーコンターのレンジを用いて表示しており、全く同じ結果が得られている。以上より、上述した並列計算手法を用いて正しく結果が得られる事を確認した。

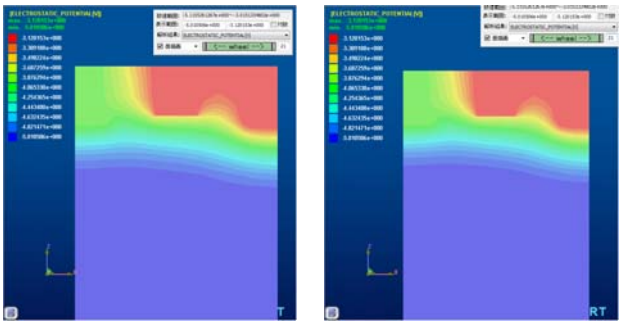

図 3　1CPU コアによる計算結果（左図）と 4CPU コアによる計算結果（右図）

3.2. 計算効率

並列計算時に、CPU コア数を増加させていくと電流連続方程式の行列計算の収束回数が著しく増加していき、並列計算の効率が悪化する問題に直面した。以下に、図 3 で示した MOSFET 計算の中で、一回行列を解くのに要する計算時間と収束回数（iteration）の CPU コア数依存性を、ポアソン方程式の求解時（表 1)、電流方程式の求解時（表 2）について示す。

表 1　ポアソン方程式ソルバーの CPU コア数に対する計算時間と収束回数

CPU コア数	計算時間[sec]	iteration
1	2.255657	42
2	2.044689	76
4	1.061839	74
8	0.642902	73
12	0.737887	94

表 2　電流連続方程式ソルバーの CPU コア数に対する計算時間と収束回数

CPU コア数	計算時間	iteration
1	44.68221	896
2	24.04335	997
4	12.42711	986
8	17.47534	2122
12	34.28479	4942

表 2 に示す通り、CPU コア数を増加させていくと、電流連続方程式の収束回数が極端に増加して計算の高速化を妨げている。この結果を踏まえ、非並列計算時の計算速度だけではなく、並列計算時には収束回数の増加をある程度抑え、計算効率が高くなる前処理と行列解法の調査と、更にパラメータ調整を行い得られた結果を以下に示す。

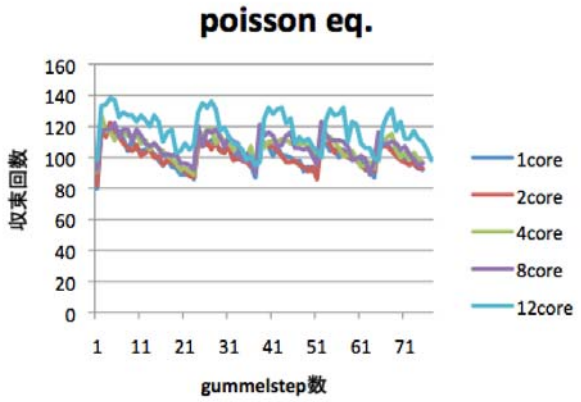

図 4　ポアソン方程式ソルバーの gummel step 数に対する収束回数の CPU コア数依存性

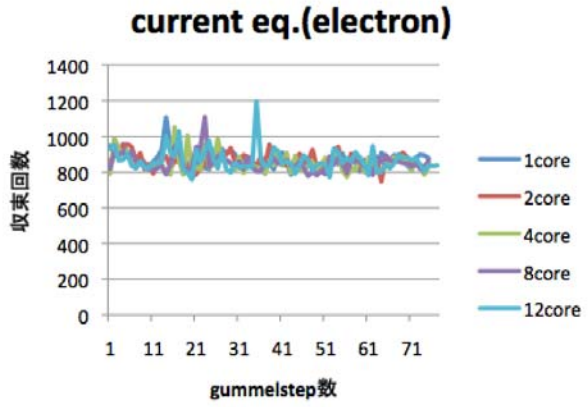

図 5　電流連続方程式ソルバーの gummel step 数に対する収束回数の CPU コア数依存性

図 4、図 5 は、修正を行った後のポアソン方程式ソルバーと電流連続方程式ソルバーの gummel step 数に対する収束回数の CPU コア数依存性を示している。修正により、CPU コア数が増加しても収束回数の増加を抑えることに成功している。特に電流連続方程式ソルバーにおいて劇的な効果が得られている。

ここまでの作業を踏まえた上で、以下に示すテスト例題を用いて並列化効率の評価を行う。

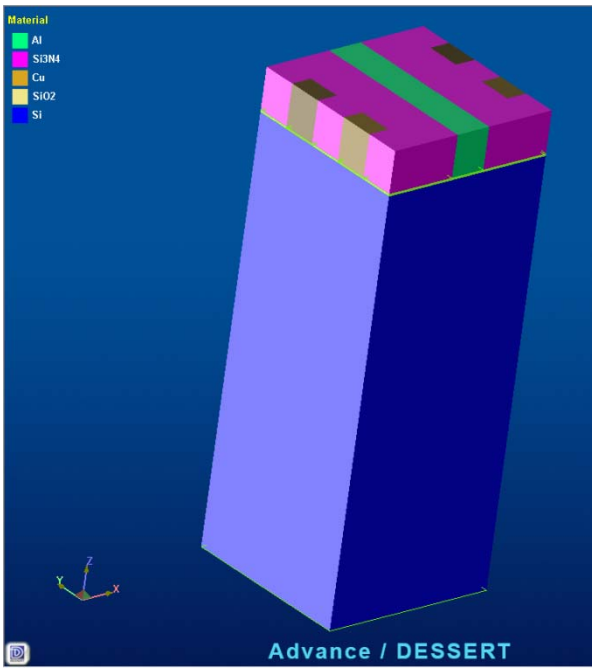

図 6　テスト例題に用いた CMOS 構造

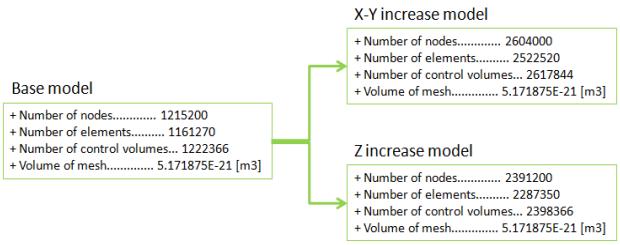

図 7　基本モデルとそこから作成した 2 つのモデル

評価に際して、基本となる節点数 100 万程度のモデルを用意し、そこから、並列計算時に最もデータ転送量が少なくなるように格子を分割した 250 万節点程度のモデル（z increase model）と、最もデータデータ転送量が多くなるように格子を分割した 250 万節点程度のモデル(x-y increase model)を作成し、２つのモデルについて並列化効率を取得した（図 8）。これにより、並列計算時

のデータ転送量による効率の違いを比較する事が可能となる。それぞれのモデルについて取得した、非並列計算時（1CPU コア）での計算時間と並列計算時の計算時間に対する割合の CPU コア数依存性を図 8 に示した。ここで、縦軸の値の逆数が並列化効率を示している。結果は、4CPU コアを超えたあたりから並列化効率に違いが現れ、z increase model の方が、x-y increase modell よりも高い並列化効率が出ているものとなった。この２つのモデルの差がデータ転送量の違いによる並列化効率の差となっている。ここでは、36CPU コア使用時に、z increase model で約 10 倍、x-y increase model で約 7 倍の並列化効率となっている。

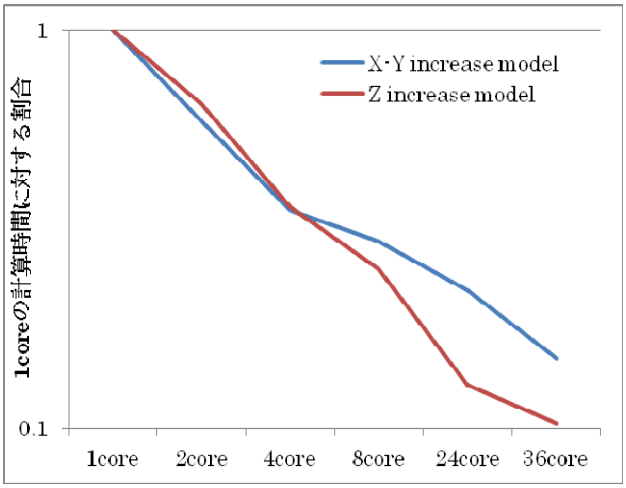

図 8　1 CPU コアによる計算時間に対する使用 CPU コア数を増加させていった場合の計算時間の割合（両対数グラフ）

ここまでのデータ取得で使用した環境は弊社のＰＣクラスター環境となっており、データ通信速度は、本格的な並列計算システムと比較すると十分ではない可能性がある。そこで、海洋研究開発機構が所有する UV2000 において取得した並列化効率を以下に示す。これによると、図 8 に比べて高い並列化効率を少ない CPU コア数で実現していることが判る。16CPU コアで約 9 倍 32CPU コアで約 16 倍の並列化効率を達成している。この違いは、上述した使用環境におけるデータ通信速度が由来となっていると考えられる。

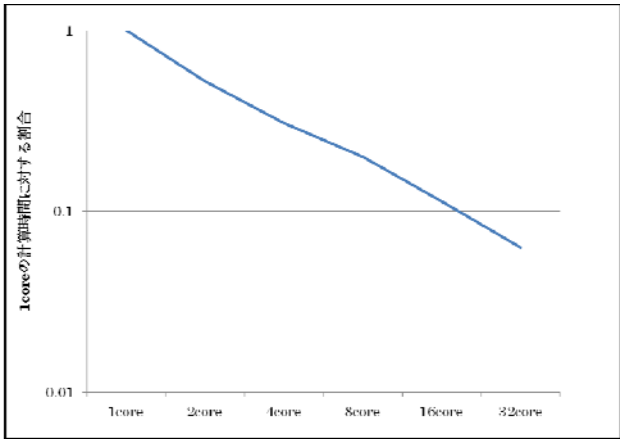

図 9　1 CPU コアによる計算時間に対する使用 CPU コア数を増加させていった場合の計算時間の割合（両対数グラフ）: UV2000 におけるデータ

4. まとめ

今回採用した並列計算手法と行列計算ライブラリを用いることにより、16CPU コア使用時に約 9 倍、32CPU コア使用時に約 16 倍の並列化効率を得る事が可能である事を示した。この並列化した TCAD プログラムを用いる事により、大規模な TCAD 計算が手軽に実行可能とする事が出来る。

参考文献

[1]　http://www.mcs.anl.gov/petsc/

※ 技術情報誌アドバンスシミュレーションは、アドバンスソフト株式会社 ホームページのシミュレーション図書館から、PDF ファイルがダウンロードできます。（ダウンロードしていただくには、アドバンス/シミュレーションフォーラム会員登録が必要です。）